(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
10 May 2002 (10.05.2002)

PCT

(10) International Publication Number
WO 02/36225 A1

(51) International Patent Classification[7]: A63F 9/24

(21) International Application Number: PCT/US01/46939

(22) International Filing Date:
2 November 2001 (02.11.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/244,795 2 November 2000 (02.11.2000) US
60/244,796 2 November 2000 (02.11.2000) US

(71) Applicant: ATLANTIS CYBERSPACE, INC. [US/US]; Bldg. 12, 874 Dillingham Blvd., Honolulu, HI 96817-4598 (US).

(72) Inventors: SCALLIE, Laurent; c/o Atlantis Cyberspace, Inc., Bldg. 12, 874 Dillingham Blvd., Honolulu, HI 96817-4598 (US). BOUTELIER, Cedric; c/o Atlantis Cyberspace, Inc., Bldg. 12, 874 Dillingham Blvd., Honolulu, HI 96817-4598 (US).

(74) Agent: CHONG, Leighton, K.; Ostrager Chong & Flaherty (Hawaii), Ste. 1200, 841 Bishop Street, Honolulu, HI 96813-3908 (US).

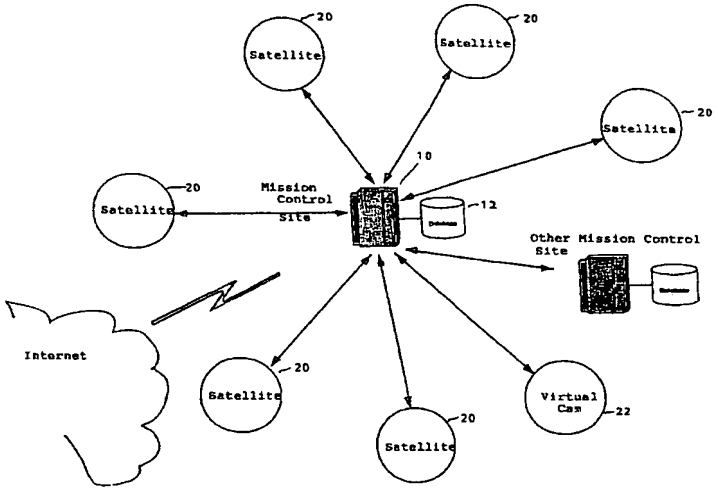
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, OM, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*
— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

[Continued on next page]

(54) Title: VIRTUAL REALITY GAME SYSTEM WITH PSEUDO 3D DISPLAY DRIVER AND MISSION CONTROL



(57) Abstract: A virtual reality game system and method uses pseudo drivers to generate stereo vision outputs for a 3D stereoscopic display from game software normally intended for output to a 2D display of a conventional game console or PC. A "mission control" system is also provided for controlling multiple game playing satellite computers (20) on a network running many game programs of different types from different publishers. The mission control program sends generic control commands to satellite computers for controlling any of the game programs, and the satellite game control program loads a game-specific command set from its database (12) for controlling the selected game program, and also provides the mission control program with information on the status of the game program. A plurality of mission control sites (10) can be connected via Internet to a network server which provides an online interface to players anywhere.

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

VIRTUAL REALITY GAME SYSTEM WITH PSEUDO 3D DISPLAY DRIVER & MISSION CONTROL

SPECIFICATION

TECHNICAL FIELD

This invention generally relates to virtual reality game systems which provide a three-dimensional (3D) immersive experience to game players, and more particularly, to methods for creating 3D stereo vision displays from popular video games, and a mission control system for administration of multiple game playing satellites (pods).

BACKGROUND OF INVENTION

Commercial virtual reality games are currently played at VR game stations with one or more players. To create an immersive environment without the high cost of installing surrounding wall displays in large room environments, the commonly used VR game station typically provides a VR game that is played by a player wearing stereoscopic goggles or other 3D head-mounted display (HMDs) and manipulating a weapon or other action equipment while executing physical motions such as turning, aiming, crouching, jumping, etc., on a platform or cordoned space. The VR games played on conventional VR game stations typically are written for the specific, often proprietary, hardware and operating systems provided by manufacturers for their VR game stations. As a result, there are only a limited number of VR games available for play at current VR game stations.

Players of VR games often want to play games that are popular video games they are used to playing on game consoles or PCs. Even though many video games are written to create 3D game effects, the common video game console or PC hardware supports image displays for 2D monitors or TV screens. While the 2D displays allow the viewer to view the image in simulated 3D space, it does not provide the immersive depth of vision of a true 3D experience. It is as if the viewer is seeing the 3D image with only one eye. Popular video games therefore are not used at VR game stations employing stereoscopic 3D displays unless the publishers of those video games have chosen to write versions for operation on the hardware and operating systems used at VR game stations of the different manufacturers.

It would therefore be very desirable to have a VR game system in which popular 3D video games written to be displayed on 2D display hardware can be operated to provide a 3D stereoscopic display

without having to re-write the video game software for the 3D display hardware. It would also be very useful for a new VR game system to enable other 3D game services for VR game players based upon popular video games they want to play on VR game stations.

Another problem with commercial game systems is that video games and VR games are commonly played at stand-alone game stations. A player typically chooses whatever game he/she wants to play and queues up at or is assigned a stand-alone game station which is loaded with the selected game. Some arcade systems have a computerized station for handling some common administrative functions such as player sign-in, maintaining player accounts, or logging game dates or selections. However, they do not utilize centralized control to keep track of and monitor plays at multiple game stations unless all the games are written in the same proprietary format used by the producer of the arcade system. If a multi-player network game is offered, the game stations are loaded only with the same game and exchange data for only that game via a local network connecting the game stations together. As a result, current arcade systems either do not offer the many popular video game titles written by other game publishers, or offer another game title on a stand-alone game station or multi-player game title on dedicated networked game stations which do not offer other game titles and are not subject to centralized control.

SUMMARY OF INVENTION

In accordance with one aspect of the present invention, a method (and system) for operating three-dimensional (3D) application software intended to provide output to a two-dimensional (2D) screen display comprises:

(a) running the application software in its normal mode to generate 3D application data output which is normally to be sent to an application programming interface (API) driver for the 2D screen display;

(b) intercepting the 3D application data output from the application software and redirecting the data to a pseudo driver for generating a 3D stereoscopic display; and

(c) using the pseudo 3D display driver to generate a 3D stereoscopic display.

In a preferred embodiment, the 3D application is a 3D video game, and the 3D stereoscopic display is a set of head-mounted stereo vision goggles used in a virtual reality (VR) game system. The VR game system employs the pseudo 3D display driver to convert 3D game data from existing 3D video game software intended for 2D screen display to right and left stereoscopic image data for the 3D stereoscopic display. Conversion to stereo vision requires the generation of specific right and left image viewpoints which are combined by human vision to yield an immersive 3D image. The Pseudo Driver converts the 3D game data output of the video game software in any of the application programming

interface (API) formats commonly used for popular video games to an API format that supports the handling of stereoscopic image outputs, thereby allowing hundreds of existing 3D video games to be played in a commercial VR game system. The invention method can also be used to generate 3D stereoscopic displays for games played on video game consoles or PCs for home use.

As a further aspect of the invention, the intercepted 3D game data can be stored by a 3D data recorder for later play back. In this mode, a game player can replay a game or scene of a game they previously played, or another player can re-enact the game played by another player. The 3D game data can also be transmitted or downloaded to a remote player through an online interface. This would allow the replay of the player's 3D visuals at home or on other hardware platforms without the original game software (like replaying previously recorded video).

The intercepted 3D game data being re-directed to the Pseudo Driver can also be overlaid, edited, or combined with other 2D or 3D images through a mixer for real-time enhancement of the resulting displayed 3D content. Examples include high-score rewards, promotional information, and logo animation before, during, and after a game or mission.

The Pseudo Driver for the 3D stereoscopic display can also be operated in tandem with other pseudo drivers such as drivers for stereo sound and/or directional force feedback.

In another aspect of the present invention, a mission control (administration) system for controlling multiple game playing satellite computers on a network comprises:

(a) a mission control computer which operates administrative programs for performing administrative functions for multiple game playing stations connected by the network;

(b) a plurality of game playing satellite computers provided at respective game playing stations each maintaining a plurality of game programs;

(c) a network connecting the mission control computer to the plurality of game playing satellite computers,

(d) wherein said mission control computer includes a mission control program for controlling the plurality of games available to be played on the game playing satellite computers by issuing generic control commands to the game playing satellite computers, and

(e) wherein each of said game playing satellite computers includes a satellite game control program for controlling each of the plurality of game programs available to be played on the satellite computer by receiving a generic control command to start a selected game program issued by said mission control computer and loading in response thereto a game-specific command set corresponding to the selected game program, and by providing said mission control computer with a status report of the status of the selected game program being played on the satellite computer.

In a preferred embodiment of the mission control system, the satellite computer control program scans game log files as games are played and extracts game status information from the log files for its status reports to the mission control computer. The mission control computer control program uses the status report information for a wide range of administrative functions. For example, the mission control computer can generate system-wide gaming reports, membership and player statistics, detailed statistics on specific games played by specific players, current status of the system, hardware, and software troubleshooting, etc.

The satellite computers each use the same control program and maintain in a database the game-specific command sets for the game programs offered on the satellite computer. The game-specific command sets are initially derived by analyzing each game offered on the system and determining the activation, control and termination logic for each game. When a generic control command is issued by the mission control computer to start a particular game, the satellite control program loads the selected game with the corresponding game-specific command set. In this manner, the mission control computer can maintain centralized control of the game playing stations while offering many different games for play.

The mission control site may be networked to another mission control site or to a plurality of mission control sites through a wide area network or the Internet. The databases of multiple mission control sites can be replicated to a master database of a network server that provides an online interface for players in any location through the Internet. The online interface allows the system to offer a wide range of related entertainment services to players anywhere, such as looking up statistics for games they or their buddies have played at any of the mission control sites, comparing their statistics to players at other sites, downloading statistics, maintaining accounts, joining groups of players, and communicating with other players.

Other objects, features, and advantages of the present invention will be explained in the following detailed description of the invention having reference to the accompanying drawings.

BRIEF DESCRIPTION OF DRAWINGS

FIG. 1A is a block diagram illustrating the overall invention method of intercepting 3D game data and using pseudo 3D display drivers for generating a 3D stereoscopic display, and **FIG. 1B** is a block diagram illustrating a preferred method for operation of the pseudo driver through the use of the "dll wrapper" method..

FIG. 2A is a diagram illustrating the conventional API function call for a 2D display from a first type of PC game (OpenGL) software, as compared to **FIG. 2B** illustrating the pseudo API call for generating a 3D stereoscopic display.

FIG. 3A is a diagram illustrating the conventional API function call for a 2D display from a second type of PC game (Glide) software, as compared to **FIG. 3B** illustrating the pseudo API call for generating a 3D stereoscopic display.

FIG. 4A is a diagram illustrating the conventional API call for a 2D display from a third type of PC game (DirectX) software, as compared to **FIG. 4B** illustrating the pseudo API call for generating a stereoscopic display.

FIG. 5 is a diagram of a virtual reality (VR) game system using pseudo 3D display drivers to drive dual graphics cards for generating a 3D stereoscopic display for different types of PC game software.

FIG. 6 is a diagram of a VR game system using pseudo 3D display drivers to drive a single dual-head graphics card for generating a 3D stereoscopic display for different types of PC game software.

FIG. 7 is a diagram illustrating a mission control system having a mission control (administration) computer connected to multiple game playing satellite computers (stations or pods) for centralized control in accordance with the present invention.

FIG. 8 is a diagram illustrating a network server connected to multiple mission control sites and providing an online interface to players anywhere through the Internet to services based on data replicated from the mission control sites.

FIG. 9 is a flow chart illustrating the sequence by which the control program at a satellite computer responds to a generic control command from the mission control program to load and operate any one of a plurality of game programs offered on the system.

DETAILED DESCRIPTION OF INVENTION

In the following description of the invention, a 3D application software generates 3D application data intended for rendering to a 2D display, but the 3D application data are intercepted and rendered by pseudo drivers for a 3D display instead. In a preferred implementation, the 3D application is a 3D video game, and the 3D display is a stereoscopic display device. The advantages of this implementation are

described in terms of the capability of configuring a commercial virtual reality (VR) game system (with multiple pods) to offer players their choice of many popular video games in an immersive VR mode with stereo vision. However, it is to be understood that the principles of the invention disclosed herein apply equally to other types of games, programs, and 3D applications, including, for example, CAD applications, simulation applications, and the like, as well as to other use environments, such as home use, standalone PCs, networked game stations, and online (Internet) gaming.

Referring to **FIG. 1A**, the basic method and system of the present invention is illustrated for playing one of many popular 3D video games that a player may want to play in 3D vision. The existing (previously written) 3D video game software 10 is played by a Player and generates a stream of 3D visuals through a game engine that outputs 3D game data. Video games are written using one of several common Application Programming Interfaces (API) for handling the rendering and display functions of the game. In a conventional mode (dashed arrows), the 3D game data (series of polygons making up image objects to appear in scenes, and light, shading, and color data) are output with API function calls to conventional API drivers 12, which render the 3D game data into display image data that are fed to a graphics display card 14 and result in a 2D image displayed on a 2D display monitor 16.

In the present invention (solid line arrows), the 3D game data output of the video game software 10 are intercepted and redirected to pseudo API drivers 20 which generate right (R) and left (L) stereoscopic image outputs to right and left stereoscopic display cards 22, 24 that generate the resulting 3D stereoscopic display on a 3D display device 26. "Stereo vision" refers to immersive visual images which provide depth perception to the viewer. Depth perception is obtained by delivering appropriate right and left offset images to the user's right and left eyes.

The API function calls intercepted and re-directed to the Pseudo API Drivers 20 result in the intercepted 3D game data output being processed to R/L image data that can be viewed on a 3D display device, such as VR goggles, helmet, or "no glasses required" 3D monitor. In order to use any of the hundreds of existing PC games, the Pseudo Drivers are written to handle the common API formats used for PC games, such as Glide (TM), developed by 3dfx Interactive, Inc., of Alviso, CA, OpenGL (TM), developed by Silicon Graphics, Inc., (SGI) of Mountain View, CA, or DirectX (TM), distributed by Microsoft Corp., of Redmond, WA.

As illustrated in **FIG. 1B**, the invention method intercepts and redirects the API function calls and 3D game data output from the existing 3D video game software 10 to Pseudo API Drivers 20. In the preferred implementation shown using the so-called "dll wrapper" method (specific examples described in detail below), the Pseudo Drivers 20 consist of a Wrapper 21 which is given the same name in the System directory as the dynamic link library ("dll") for the original API drivers ("Original Drivers"), while the

original dll is renamed under a different name and maintained with the Original Drivers. When the video game software is initialized, it calls the dll for the API drivers in its usual mode. Due to its assumption of the original dll name, the Wrapper 21 is called instead of the original dll and drivers and effectively intercepts the API function calls and 3D game data of the video game software. The Wrapper 21 establishes a Stereo Viewpoints module 22 and sets up parallel R and L rendering engines from the renamed original dll and drivers, one rendering engine 23 for rendering right (R) image data, and the other rendering engine 24 for rendering left (L) image data. The Wrapper 21 sends the 3D game data to the Stereo Viewpoints module 22 where right (R) and left (L) viewpoints are calculated or specified for the 3D game data, resulting in R View data and L View data. The API function calls are directed by the Wrapper 21 to the R rendering module with the R view data, resulting in rendering the R image data, and to the L rendering module with the L view data, resulting in rendering the L image data. The R and L image data are then sent to the R and L display cards for the 3D stereoscopic display (see **FIG. 1A**).

In the invention, the Pseudo Driver intercepts the 3D game data between the game and the API. The 3D game data can thus be rendered into stereo vision for any specified viewpoint. In the conventional mode by contrast, the data stream from the game goes to the API which is specific to the video card, and undergoes rendering and transformation to an image fixed as 2D. The Pseudo Drivers of the invention method intercept the game data stream and invoke the same (or comparable) rendering functions to render the 3D game data into 3D stereoscopic image data, by generating specific right and left image viewpoints. The right and left image data are sent as outputs to the display cards 22 and 24, which then generate the respective bit-mapped image outputs to activate the display elements in the corresponding right and left eyepieces of the stereoscopic display unit 26. In the preferred embodiment shown, two separate display cards are used for the two stereoscopic image feeds for greater processing speed and throughput.

Computational methods for generating right and left stereoscopic images from given image data are well known, for example, as described in "3d Stereo Rendering Using OpenGL (and GLUT), by Paul Bourke, November 1999, available at the Internet page http://astronomy.swin.edu.au/pbourke/opengl/stereogl/. The method of determining the right and left eye offset and computing corresponding left and right eye images is deemed to be conventional and not described in further detail herein.

Referring again to **FIG. 1A**, an integrated Pseudo Driver system can also include a 3D game data recorder 30 (3D Recorder) for storing the 3D game data for later playback, and a mixer 40 for enhancing the 3D content, such as by overlaying, editing, or combining with other 2D or 3D images. The 3D Recorder 40 records the 3D game data stream (vertices, polygons, textures, etc.) for subsequent playback without the need to re-access the game software, such as for providing visuals while debriefing players

after a game session or for replaying for a player's personal use. The mixer 40 allows other images, 2D or 3D, to be mixed or interspersed with the game images. For other 3D content, the mixer 40 takes the form of a dual rendering module which renders the other 3D content and combines it with the game content. It is advantageous to record 3D game data with the 3D Recorder between the Pseudo Driver Wrapper and the mixer (dual rendering module), because all API types will have been converted into the chosen 3D image data format (DirectX 8, as explained below). Using data compression techniques, the large amount of data can be minimized and stored to disk. The data stream can be played back by simply sending it to the dual rendering module. If the data stream is sent to the 3D Recorder between the game and the Pseudo Drivers, then the game data can be played back simply by sending it to the corresponding API.

Because of the separation between the Wrapper 21 and the mixer (dual rendering module) 40, the mixer can always be running. This allows the system total control of the display at all times, and avoids any lapse in the display if, for example, control is switched to another game. When the next game is run, the API Wrapper called by the new game re-connects with the dual rendering module.

Use of Existing Game Software in VR Systems

State-of-the-art first person games are composed of a "game engine", an object-oriented scriptable logic, and game "levels". The game engine is the essential technology that allows for 3D graphics rendering, sound engine, file management, networking support and all other aspects of the core application. The content of the game sits on top of the rendering engine in the form of scripts and levels basically setting up the series of scenes and actions ("world map") forming the visual environment and the logic within it.

Tools provided by game developers are available for modification of the scripted logic of the various objects in the world map, as well as generation of new environments. For PC games, this allows for new content to be created by game developers and the life cycle of the game to be significantly greater. The current trend in game development is to license a specific game engine and allow game developers to focus on content creation, the concept and implementation of the levels, sounds, models, textures and game logic. Using those editing tools, customized game environments can be produced, characters created, weapons and game objects designed, and special game logic implemented to create the desired game content.

Conventional 3D video games are written to be run on conventional hardware and operating systems for display on a 2D monitor, and thus the conventional experience is basically 2D. The 3D game data executes function calls to conventional API drivers for the game that result in a 2D screen image being generated. The conventional game system renders a 3D scene as a centered 2D image as if the

user were viewing it with one eye. It is desirable to use existing 3D games for play in VR systems that engage players with a 3D stereo vision display for a more immersive game experience. Since the existing games output 3D game data, the 3D game data can be converted to a 3D display. However, mere connection of a 3D monitor to a standard 3D game like Quake3 (TM), distributed by Activision, Inc., _____, CA, would not yield a stereo vision image. Doubling a centered image using 3D display hardware also would not yield a stereo image. Only the generation of specific right and left image viewpoints for stereo vision will yield a correct stereo image on a 3D display unit.

3D display technology includes, but is not limited to, HMDs, no-glasses-required monitors, LCD glasses, and hologram display units. Typically, all of these hardware types require two separate 2D input images, one for each eye. Each new type of 3D display technology comes with its own 3D format. Typically, they conform to one of the following standard formats (from highest quality to lowest quality): separate right and left (R/L) images; frame sequential images; side-by-side (left/right) images; top-and-bottom (over/under) images; or field sequential (row interleaved) image signals.

The highest quality stereo vision signal is simply two separate R/L image signals. The remaining methods use some method of compression to pack both left and right signals into a single signal. Because of this compression, and overloading of a single signal, the stereo vision image quality is lowered, and/or the frame rate is lowered. The lower quality, "single signal" methods are typically used by lower-priced stereovision hardware, like LCD glasses. Some hardware vendors, such as nVidia Corp., of Santa Clara, CA, have recently provided support for single-signal, stereo vision formats. For example, the nVidia stereo vision drivers are contained within the nVidia video card-specific driver, nvdisp.drv. The nVidia driver effectively converts a 3D game written for DirectX or OpenGL to be viewable in stereo vision using any single-signal 3D device that is connected to the nVidia video card. However, these card-specific drivers only work if the manufacturer's video card is used. Conventional hardware manufacturers do not support card-independent high-end, separate right and left image signals.

Another important aspect of the invention is the interception of the data stream at the game-API level. Conventional stereovision drivers are established between the API and the video card, and the code existing between the API and the video card requires hardware-specific code. Drivers on that level need to be made by the manufacturer of the video card hardware, which is a drawback in a game system that offers many different games using the same video card hardware. Another drawback is that the data has already undergone a 3D game data to 2D image data transformation, and is therefore fixed as 2D. Once the data are converted to 2D, the 2D data can be converted to stereovision only with "less visually accurate" mathematics.

In the preferred embodiment of the invention, two separate video cards 22 and 24 are used for

the separate right and left signal inputs of high-end 3D display devices. Doubling the number of video cards allows for the right and left stereo image to be rendered separately and simultaneously. This avoids the typical 2x slowdown required to display stereo rather than mono. The Pseudo Driver thus allows a normal 3D game to power two video cards, which in turn can power high-end 3D display hardware such as V6 or V8 (TM) Stereovision Head Mounted Displays, distributed by Virtual Research Systems, Inc., of Santa Clara, CA, Visette (TM) Stereovision Head Mounted Display, distributed by Cyber Mind, Ltd., of Leicestershire, UK, Datavisor (TM) Stereovision Head Mounted Display, distributed by N-Vision, Inc., of McLean, VA, or DTI 2015XLS or 2018XLQ (TM) Stereovision Monitor, distributed by Dimension Technologies, Inc.

Pseudo 3D Display Drivers

In the present invention, the 3D game data output of existing game software are intercepted and re-directed to Pseudo Drivers for 3D display in place of the conventional API drivers for 2D display. The Pseudo Drivers execute the same or comparable image rendering functions but generate the specific right and left image viewpoints required by 3D display devices. The Pseudo Drivers only convert the 3D game output of the game software and do not affect or manipulate the game software itself. Thus, the Pseudo Drivers can produce a 3D display from conventional 3D game software without requiring access to or modification of the game source code.

3D display technology has developed to offer very high resolution and wide field of view. When used with a head mounted display unit (HMD) which allows direct head tracking, VR systems can offer a very immersive virtual reality experience for the player. Other 3D display devices that may be used include 3D monitors, such as the DTI3D (TM) monitor distributed by Dimension Technologies, Inc., of Rochester, NY, which delivers a stereo vision image without requiring the use of stereoscopic glasses. Most new 3D display technology can be hooked up to games running on standard Intel-based PCs with the Microsoft Windows (TM) operating system.

Typical graphics API's have some 400 functions that the game program can call to render 3D polygonal scenes. These functions, generally speaking, have names like LoadTexture, SetTexture, RenderPolygons, Display Image, etc. All of the API's functions are held in a dynamic link library (dll). The API's .dll is stored in the computer's C:\Windows\System directory. Depending on which API format it is written for, a game will automatically load the appropriate .dll stored in the System directory, and the functions contained within are used to render the game's 3D world map to the 2D screen. The API converts the data internally, and forwards the data to the video card-specific driver. The driver optionally modifies the data further into a format specific to the current video card hardware. The video card renders the data to the screen in the form of textured polygons. The final image appears on the user's monitor as a 2D projection of the 3D world map.

The Pseudo Driver of the present invention intercepts the data being sent from the game to the API. The simplest method to do this borrows from a technique called ".dll wrapping". In this method, a "Pseudo API" is named and substituted for the usual original API for which the game issues the display function calls. That is, the Pseudo API assumes the identity of the usual API's .dll that the game is looking for. This substitution is done at the installation level for the VR system by storing the Pseudo API in the System directory in place of the original API. When the game executes function calls for the API, the Pseudo API is called and intercepts the 3D game data. The data stream between the game and the rendering API consists of thousands of vertices, polygons, and texture data per frame. The Pseudo API then either executes calls, or issues subcalls to the original APIs which are set up to be running in the background, for the usual rendering functions, then passes the rendered data to the Pseudo Driver matched to the type of 3D display unit used in the VR system. The Pseudo Driver generates the card-independent R/L stereoscopic image signals which are passed as inputs to the 3D display unit.

Example: Pseudo OpenGL Driver

Many popular PC games are written for OpenGL API, such as Quake3. As illustrated in **FIG. 2A (Prior Art)**, the game run in conventional PC-based mode initializes with an API call for the OpenGL dynamic link library, called "opengl32.dll", stored in the C:\Windows\System directory. The game software loads the opengl32.dll, then sends the stream of game data generated by play of the game to opengl32.dll for linkage to the appropriate API drivers for rendering the game's series of scenes to the 2D screen. The API drivers render the game data to image data and sends the image data to the graphics card used by the API to drive the 2D display.

As shown in **FIG. 2B**, a Pseudo OpenGL Driver has a wrapper named "opengl32.dll" is substituted in the System directory in place of the OpenGL .dll formerly of that name. When Quake3 is run, it calls for the "opengl32.dll" and binds with the Pseudo OpenGL Wrapper that was substituted. In this case, the OpenGL API is never actually initialized; in fact, it is not needed on the machine at all. The Pseudo OpenGL Driver linked to the psuedo OpenGL wrapper pretends to be the OpenGL driver, however, all the data sent to it is converted into a format that can be rendered for stereo vision by a dual rendering system for the dual R/L stereoscopic image outputs. DirectX 8 is used as the rendered data format since it can support the use of multiple outputs to multiple graphics cards. For about 370 functions, some translation and/or redirection is required. Generally speaking, only about 20% of the functions are actually used by games. Each of these functions has a small amount of code that is translated. Translation could be as simple as calling "LoadDirectX8Texture", when "LoadOpenGLTexture" is called, for example. The DirectX 8 calls are linked through the real DirectX 8 .dll ("d3d8.dll"). Other functions require large amounts of code that converts vertex, index, or texture data. All the game data is handled in this way by the Pseudo Driver. The Pseudo Driver effectively ports Quake3 for OpenGL and 2D display to DirectX 8 for stereoscopic display without touching Quake3 source code.

Example: Pseudo Glide Driver

The Glide API has been used in many popular games, but is no longer being supported. A Glide-only Pseudo Driver was created for use only for Glide games. Glide is technically unusual in that it allows access to multiple graphics cards (but only if 3dfx cards are used). This made the creation of the Glide Pseudo Driver easier than for OpenGL which does not allow access to multiple video cards. As before, **FIG. 3A (Prior Art)** shows a Glide game run in conventional mode for a 2D display, and **FIG. 3B** shows the Glide game run in pseudo wrapper mode for a 3D display. Source code for a Pseudo Glide2x.dll wrapper was written, and stored in the C:Windows\System directory. The Pseudo Glide wrapper exports the same rendering functions as the real Glide2x.dll. From the outside, the two .dlls are indistinguishable. As a result, when a Glide game such as Unreal Tournament is run, it loads the Pseudo Glide2x.dll from the C:Windows\System directory. The game Unreal Tournament then sends game data to the Pseudo Glide wrapper, which manipulates the data, changing it into a format for stereoscopic display to two video cards for the right and left image viewpoints.

Example: Pseudo DirectX Driver

Many popular games today, such as Unreal Tournament, are written for DirectX 7 or earlier versions or have the option to render using DirectX 7. The Pseudo Driver system is set up to use DirectX 8, because DirectX 8 can support multiple hardware devices. Therefore, games written for DirectX 7 uses a pseudo wrapper which provides for conversion from DirectX 7 to DirectX 8. Games written for DirectX 8 can use a pseudo wrapper which links to the real DirectX 8 functions and the required further links for generation of the R/L stereo vision outputs.

DirectX uses a linking structure named Common Object Method (COM), which is a different method of storing functions inside dynamic link libraries. Therefore, the Pseudo DirectX wrapper was written to handle the COM link structure. The code for the DirectX COM wrappers is more complex than the OpenGL, or Glide wrappers. For example, in the opengl32.dll structure, all of the rendering functions are accessible to OpenGL programmers. However, the DirectX COM structure has an initial index which only points to 3 categories of functions, as follows:

ValidatePixelShader
ValidateVertexShader
Direct3DCreate8

The category index has a link structure which points to the actual rendering functions one layer deeper. When a DirectX 8 game initializes, the DirectX API named "d3d8.dll" is loaded. The game must first call the Direct3DCreate8 function, which returns a class pointer. This class pointer can then be used to access all DirectX 8 rendering functions. Thus, in addition to the standard .dll wrapper, the pseudo DirectX wrapper handling the COM method also requires a wrapper for the class.

FIG. 4A (Prior Art) shows a DirectX game run in conventional mode for a 2D display, and **FIG. 4B** shows the DirectX game run in pseudo wrapper mode for a 3D display. Source code for a Pseudo DirectX wrapper was written, and stored in the C:Windows\System directory. There are actually two DirectX wrappers stored, one for the DirectX 7 .dll named "d3dim700.dll" for games written for DirectX 7, and one for DirectX 8 .dll named "d3d8.dll" for games written for DirectX 8. The pseudo d3dim700.dll converts DirectX 7 function calls and data into DirectX 8 function calls, whereas the pseudo d3d8.dll links directly to DirectX 8 function calls. The Pseudo DirectX wrapper renders the game data into a format for stereoscopic display to two video cards for the right and left image viewpoints.

Integration of Pseudo 3D Display Drivers in VR Game System

Referring to **FIG. 5**, an example of the virtual reality game system is shown incorporating pseudo 3D display drivers for existing PC games to generate a stereo vision display. The system can accommodate most of the popular games that are written for OpenGL, DirectX 7, and/or DirectX 8. Pseudo OpenGL, DirectX 7, and DirectX 8 wrappers take the 3D game data output of any of the games and re-directs them to Dual Rendering links to real DirectX 8 rendering functions. The resulting R/L stereo image outputs are fed to dual graphics cards, which are nVidia GeForce2 cards using the card-specific driver nvdisp.drv in this example. The separate R and L image display outputs are fed to the respective R and L eyepieces of a stereo vision head mounted display. A parallel system can be configured for Glide games using a Pseudo Glide wrapper and Glide-specific graphics cards.

FIG. 6 shows an alternate configuration in which the R/L stereo image outputs are fed to a single dual-head graphics card, which is an ATI Radeon 8500 Dual Monitor card in this example. The single "dual head" card has 2 VGA monitor-capable outputs. Some of the cards components, like the PCI interface for example, are shared between the two devices. As a result, the single card solution is slower than the same system with dual cards. Thus, the dual-head system offers a tradeoff of somewhat lower performance against a lower cost than the two-card system.

Extremely high-end stereo devices take two inputs, one for each eye. Typically, the two inputs are provided from two separate video cameras to achieve stereoscopic vision in the final 3D display. In the invention, the Pseudo Driver instead provides a high-end synthetic connection to the 3D display through the re-direction of 3D game data to dual rendering functions and dual graphics cards to provide the two stereoscopic images. Each card (or card head) renders a slightly different image, one from the viewpoint of the left eye, and the other from the viewpoint of the right eye. Because both frames are rendered simultaneously, the typical 2x stereo vision slowdown is avoided. This allows regular PC games like Quake3 to be viewable in stereo vision using the latest 3D display technology.

The pseudo driver methodology enables an integrated VR game system to be played for most of the popular PC games known to players, and allows integration of related functions that can take advantage of its game data interception and dual rendering functions. Some of these system integration features and advantages are described below.

Pseudo Driver Architecture: The pseudo driver software architecture allows interfacing of VR input and output devices to a 3D game application without its source code. Pseudo drivers are drivers or applications that lie between the game application and the actual legitimate video, sound, input or output driver. The VR system wraps existing applications with a series of pseudo drivers.

Generic (Game-Independent) Stereo Vision Display: The pseudo driver method generically allows creating a quality depth perception display from any 3D application without needing to access its source code and regardless of the API (Glide, DirectX, or OpenGL). The outputs are two separate high quality VGA signals with no compromise in frame rate or resolution. It is not an interlaced output.

Generic (Game-Independent) Recording Engine: Since the 3D Recorder records 3D game data output from any Glide, DirectX, or OpenGL applications, it can replay the visuals of a player's game in high quality 3D vision without needing to run the original application. One of the further advantages of this is that the recorded data can be replayed on any DirectX capable player, making it possible to use an online interface allowing members to download their mission replay to their home hardware platform.

Generic (Game-Independent) Video Overlay Graphics: By leveraging the architecture of the pseudo driver, it becomes possible to fully control and even enhance the 3D game output though a mixer. The game visuals can be overlaid with other 3D content or animation, text and graphics in real time. Examples include high-scores, promotional information, and logo animation before, during or after a mission.

Native Head Tracking Support: The pseudo driver methodology allows PC games to be played as VR games using head-mounted devices. The HMDs allow for head tracking in real-time inside a game environment with 3 degrees of freedom (looking up/down, left/right and tilting) without access to the game source code. Native versus mouse emulation tracking allows for zero lag and high-resolution positioning, ultimately increasing quality immersion and reducing motion sickness. A critical benefit of native tracking is that the user does not experience head recalibration since horizontal in real space is known by the device in this mode only.

Duo Tracking Support: Use of HMDs frees up the player's hands to control a weapon or other type of action device. Currently, 3D consumer games only support a single 2 degrees of freedom input device

(mouse, trackball, and joystick). The VR system can support two 3-degrees-of-freedom tracking devices via a combination of external drivers and game script modification. In duo tracking (head tracking and weapon tracking), a player will be able to look one way and shoot the other for example.

Peripheral Input Engine: This tool enables the system to interface a variety of input devices like guns, buttons on the POD, pod rail closure sensors and the like to the 3D game or the mission control software.

Pseudo Sound & Force Feedback Drivers: A pseudo sound and/or force feedback driver can be added in tandem with the pseudo 3D display driver. This would allow real-time filtering of sounds and generating accurate force feedback for custom designed hardware like a force feedback vest. The vest can have a number of recoil devices that would be activated based on the analysis of the nature and impact locations of ammunition in the virtual opponents. For example, a rocket contact from the back would trigger all recoil devices at once, while a nail gun hitting from the back to the front as one is turning would be felt accurately by the user. Further applications of the pseudo driver method could include an intercom localized in the 3D environment and replacement or addition of game sound with other sounds.

In another aspect of the invention, principles for centralized control of multiple game playing stations are explained using the preferred example of a commercial level, multi-station game system offering many different game programs for play. Details of the system are based on a commercially available product for game arcade systems offered by the assignee of the present invention, Atlantic Cyberspace, Inc., Honolulu, Hawaii, which is referred to herein by its tradename designation *Atlantis OS*. It is to be understood that other variations and modifications may be made given the principles of the invention described herein.

Referring to **FIG. 7**, an administration or "mission control" site employs a mission control computer 10 connected by a network to multiple game playing satellite computers 20 installed at respective game playing stations or "pods". The mission control computer operates a variety of administrative programs for performing administrative functions for the game playing stations on the network. Each satellite computer operates a plurality of game programs (video games, virtual reality games, simulation games, etc.) which a player may select from. The mission control computer has a mission control program for controlling the games played on the satellite computers. The satellite computers have a satellite game control program which responds to generic control commands issued by the mission control computer to start a selected game by loading the selected game along with its game-specific command set and sending status reports to the mission control computer. The mission control computer can thus maintain centralized control of the games played on the game playing stations without having to control each of the many different games offered for play.

The Mission Control Program used by the mission control computer is the heart of the mission control system. It determines what satellite computer should be playing which games, how long the games will last, how much the player will be charged, etc. The Mission Control Program connects to each Satellite Control Program over the network to control what that satellite computer will do. The commands it issues to the Satellites can include "Start new network game", "Debrief game", "Quit current game and start new game", etc. The Mission Control Program can also connect via a wide area network or the Internet to another mission control site or to a central network server which provides an online interface to players anywhere (described in further detail below).

The mission control computer maintains in its associated database 12 game statistics, game playing data, and other information for a wide range of administrative functions. For example, the Mission Control Program can provide the operator of the system with reports, targeted marketing and promotional materials, membership details, current status of the system, hardware and software troubleshooting, etc. It can also generate and print detailed player statistics after each mission has finished as well as provide information access through the on-line interface. It can also maintain a history of all player information on the system, including: available funds on account or passcard; mailing and email addresses; current ranking among other players; etc. This information provides the system operator the ability to tailor a marketing campaign to the specific individuals based on the information stored in the database.

Referring to **FIG. 8**, the mission control system can be extended to multiple mission control sites connected via a wide area network or the Internet. The game data from each mission control site are replicated to the central server's master database that contains all the information for all of the sites on the system. The server provides an online interface that allows players anywhere to access the game data from all (participating) sites remotely. For example, the online interface can allow a player or players to view their current stats on the system from the Internet, including player history, player information, etc. This creates content on the Internet for a system website that supports interaction and communication among players anywhere. The players could also have the ability to download data from the games that they have played for their personal interests. The online interface can also allow players to maintain their handles (user IDs) or accounts in the system, change address information, create or join groups (teams or clans), and chat with other players. Individual site operators can customize their presence on the online interface depending on what player information they choose to share with the entire online community.

The Mission Control program has the ability to control multiple Satellites running different games at the same time. In conventional systems, only a particular game can be played on a stand-alone station, or the same game must be played on all designated stations offering a particular network game. In the invention by contrast, each Satellite can offer many different games, and its Control Program will load the game-specific command set from its database required for a selected one of the many games. In this

manner, Mission Control can issue a generic control command to the Satellite, e.g., to "Start (particular) game", and the Satellite then loads the game-specific command set that will enable control of that game.

An example of some of the functions performed in starting and ending a game and the commands issued by Mission Control and responses of the Satellite in the *Atlantis OS* system offered by Atlantis Cyberspace, Inc., is shown on **Table I.**

To report the current status of a game, the Satellite game control program can read the log files, the current dialog boxes or windows opened by the application (game program) running on the system, messages from the Notification API, or some other method used by the game program for external communications. For games that maintain log files, the log files can be parsed for information, e.g., whether the game is still running, when a player dies, when a player kills someone, when the game is over, when the game started, etc. By gathering this information, a status report on the game can be provided by the Satellite to Mission Control.

An example of a typical log file for the popular game "Quake" is shown in **Table II.** The log file is parsed for keywords identified by the system as providing status information, such as "version", "I am (player)", "playing demo", "exited (level)", "game over", etc. In **Table III,** examples of conversion of messages parsed from the log files into reports issued by the Satellite to Mission Control are illustrated.

The Mission Control program applies generic logic to communicate to the Satellites, however there is some minor logic based on the different types of games to control it better. For example "Quake" and "Unreal Tournament" behave slightly differently on the game server. Therefore, there is some special logic for games like "Quake" that provide full logging functionality on their servers as compared to "Unreal Tournament" which provides no logging on the server, but is provided via a "virtual cam" recorder.

In the preferred implementation, Mission Control is only notified by the Satellite when there are changes to the system state. For example, if the game were to finish then the Satellite would send Mission Control the command GAME_OVER. To prevent the assumption that a machine is working, Mission Control also expects a GAME_READY command from the Satellite every 10 seconds or so with a parameter that it is ready or not. This feature is mainly used for situations that are beyond the control of the Satellite or Mission Control where the machine locks up or the operator exits the program unexpectedly.

The Notification API are tools that allow Mission Control to determine the current status of a game program without doing any detailed research on the program itself. This is achieved by giving the game developer or program developer a set of APIs that they call in their application to notify of the current

status of the program. For example, these commands can include "Game started", "Game over", "Program started", "Player joined", "Player killed", etc. The Notification API are stored as a set of DLLs available for each operating system supported by Mission Control.

Similarly, the Control API allows Mission Control, and more specifically, the Satellite the ability to control a current game or program without knowing any detailed information about the program itself. This is achieved by providing the game developer or program developer a set of APIs that they use to issue commands that "hook into" a particular game, e.g., "Start a new game", "End current game", "Join network game", etc.

To port an existing game into Mission Control without having used the Notification API or the Control API, the game or program must be analyzed for details of the logic sequences needed to control the program, e.g., how to start and stop the game, how to tell if the program is still running, how to start the program, how to start a game server, how to join a game server, etc.

Popular games offered by different game publishers often employ different command structures and use different command protocols to start, stop, and control games. In the invention system, the command architecture for each game offered by the system is analyzed and the appropriate game control signals matched to the activation, termination, and control logic for each game are stored in the Satellite database. For example, the game control signals may be stored in a relational database indexed to each particular game. When a command is issued by Mission Control to start a particular game, the relational database retrieves the set of all command signals used by that specific game and loads it with the Satellite operating system so that activation by the player of the hardware buttons and other controls results in the correct signals being delivered to the game.

As a specific example, to control a PC-based game like "Quake", distributed by Activision, Inc., _____, CA, the computer system must send keyboard commands to the game window using the SendMessage and PostMessage commands of the Microsoft Windows (TM) operating system API functions. However, a network game like "Unreal Tournament" uses http commands for the game server, but requires Keybd_Event commands for the client station to control the game. Thus, the correct signals required for each of the game programs must be determined based upon the game's command architecture, e.g., keystrokes, http commands, TCP/IP commands, writing files, its control API, or via serial communications if there is a modem or a COM port on the computer. The Satellite Control Program can then interpret inputs from the player on the hardware console or network game commands sent from other stations via the network through an http interface into the correct signals required by the game to control its actions.

An example of typical commands for games maintained in a Command database is illustrated in **Table IV**. To let the "Unreal Tournament" game know that the player wants to shoot a weapon in the game when the player presses a trigger on the hardware console, the computer system must have the game-specific command set loaded so that a lookup of "Fire" in the command set results in the corresponding Keybd_Event of the <CTRL> signal being sent to the game. Similarly, the input "Move Forward" for this and other games will result in the <UP ARROW> keyboard signal being sent, etc.

In **FIG. 9**, a typical "Start game" sequence by which the Satellite Control Program interacts with the Mission Control Program is illustrated. At block 30, Mission Control sends a generic "Start Network Game" (Unreal Tournament) command to the Satellite Control Program. The Satellite Program reads from its database the corresponding Key Table at block 31, and Message Table for parsing the games log files at block 32. At decision 33, it tests whether Unreal Tournament is already running, or at decision 34 whether another game is running . If another game is running , it is shut down at block 35, then the parser for Unreal Tournament is initialized at block 36, and the Unreal Tournament game is launched at block 37. Once the game is ready (decision 38), the "Ready" status is reported to Mission Control at block 39. When a command is received from Mission Control to "Join network mission" at block 40, the Satellite Control Program checks whether its system is connected to the Network Mission (communicating with the other Satellites having players participating in the same mission) at decision 41, then sends a "Connected" message to Mission Control at block 42. It then checks whether Mission Control has sent the "Everyone Ready" message at decision 43, then starts the game at block 44. The game is then played by the player on the Satellite, in conjunction with other the other participating players on other Satellites, at block 45. All signals exchanged between networked players are handled at the game application level. When the "Game Over" message is parsed from the game's log files, at block 46, the Satellite then sends a "Game Over" message to Mission Control, at block 47.

Through the use of generic control commands to the multiple Satellites running many different games at different times, the Mission Control system provides a universal intermediary between the game programs and the hardware interface. Mission Control maintains data on the games played on the system by tracking the status reports of the Satellites. It can also handle other desired management functions, such as cash dispenser, briefing/training, monitoring game hardware, mission control, back-end information management, member management, Internet connectivity for remote management, automated updates, and player interactions. The Mission Control system allows the site operator multi-faceted control and interactivity with participants, thus enhancing player experiences and the value delivered.

The Mission Control system leverages a database-driven client-server architecture. Each client is fully controlling the activity of all applications and activity on the client computer and reporting to Mission Control, the central server that coordinates and monitors every single event in the system. The server

provides a comprehensive graphical user interface for monitoring full system activity and for modifying or creating individualized missions based on a variety of player-desired parameters. By developing the entire controlling software around a high-end database engine (e.g., Sybase), the Mission Control system gains the benefit of storing all information and events into a scalable database engine, the capacity of replication for backup, remote management, site link-up and on-line interface capabilities.

For purposes or playback or debriefing the players, the Mission Control system can also record the experiences of network game players by running a satellite client of the game as an "observer". The "observer" can enter the game space along with the other players and see (record) the game action. The recording can be done from different camera angles or points of view. In **FIG. 7**, this "observer" is referenced as Virtual Cam 22. The Virtual Cam is programmed to execute a sequence of views or to record specific actions in the game when certain conditions are detected.

The Mission Control system can be combined with other technologies in a totally integrated game entertainment system. The integrated system can include other 3D and VR enhancement tools like the use of player and spectator videocams to record different viewpoints using 3D space analysis, artificial intelligence methods, and a library of pre-defined camera angles and motions to generate high-quality Hollywood-style coverage of action within a virtual world.

The entire VR system can be controlled and monitored by a user friendly GUI. All user and system activity can be simply monitored. All software components can be protected via a variety of security methods including preventing decompilation of code, encryption of network packets, etc.

The online interface on the Internet provided by linking mission control sites to a network server and master database allows players to engage in enhanced services and the ability to communicate with each other and play network games. Two or more Mission Controls can combine game controls and player input and output into a format that can be streamed over the network or Internet for playback on any other system. Through the online interface, players can remotely view and download their stats, their buddies' stats, clan stats, find out who the competition is, vote for best player, change handles, email or street addresses, choose player skins or faces for games that they wish to play, etc.

The accumulation of player information at each Mission Control site will allow an operator or groups of operators to organize and automate the management and marketing to players for any entertainment site of chain of sites. For example, repeat business can be automatically targeted with email messages to existing members, or to new members introduced by existing members, competitions and special events can be organized for players via email, and past members can be automatically contacted for return visits. Using briefing and de-briefing tools, the system can store player information

on selection of games, teams, choice of bots, weapons, etc. A statistics engine can analyze and print players stats as well as store them for the on-line interface. This provides the player with a detailed account of their game.

Video conferencing capability can be provided between mission control sites to allow players to join a game at another site and to have a briefing and debriefing with other players to exchange tips and strategies, etc. When the game is over, they will be able to talk and see each other during the playback of the debriefing

The Mission Control system can also provide players' "team leaders" with access to a mini version of the Mission Control to allow them to view the progress of games from different camera angles and help with strategy development and execution. This will also increase spectator interest as they can gain insight into the leaders' methods and approaches, similar to being able to hear a coach discuss strategy while watching the game on television. A similar mini version of the Mission Control can be provided for "directors" to mimic the control center for a televised sports event where the director has access to many cameras and directs which angles the cameras focus on and decides which ones to utilize.

Thus, the Mission Control system can greatly facilitate social interaction among players and teams and excitement over the games. All the above-mentioned features can combine to provide the best possible entertainment experience from the stand point of technology, game play, social interaction, competitive aspect, spectator sports, length of the experience versus cost and ultimately lead to high repeat business and a viable financial model.

It is understood that many modifications and variations may be devised given the above description of the principles of the invention. It is intended that all such modifications and variations be considered as within the spirit and scope of this invention, as defined in the following claims.

TABLE I

Example function to perform	Command to Send to Satellite or Mission Control	Parameters to send to Satellite or Mission Control
Atlantis OS Mission Control tells the ***Atlantis OS Satellite*** to start Unreal Tournament	LOAD_GAME	None
The **Atlantis OS Satellite** determines if Unreal Tournament is currently running or if another program is running. If another program is running it will shut that down based on its knowledge of that system and then start Unreal Tournament.	QUIT_GAME[1]	None
The ***Atlantis OS Satellite*** notifies ***Atlantis OS Mission Control*** that is starting Unreal Tournament	GAME_STARTED	None
When Unreal Tournament starts the ***Atlantis OS Satellite*** then notifies ***Atlantis OS Mission Control*** that Unreal Tournament is up and running.	GAME_READY	
Atlantis OS Mission Control now tells the ***Atlantis OS Satellite*** to join a network game that has been created on another system on another ***Atlantis OS Satellite.***	NETWORK_GAME	Game #, World, Mode, Map, Host Name, Host IP, Player Handle, Skill, Blue Red & Bots
With the system has joined the game the **Atlantis OS Satellite** sends the message to **Atlantis OS Mission Control** informing it that it has joined the game.	PLAYER_CONNECTED	
During the game as a player completes an objective, kills someone, or gets killed themselves the **Atlantis OS Satellite** records the information and displays it on the LED while providing feedback to the user by the gun or vest.	Information is only sent to an LED if attached to the system	Text to send ie: "Player Name" is in the lead
When their allotted mission time ends ***Atlantis OS Mission Control*** tells the ***Atlantis OS Satellite*** to quit playing the current game.	PLAY_DEMO	
The ***Atlantis OS Satellite*** notifies the player that the game is over and please remove their headset and to open the pod rails in the case of a pod based game.	Audio commands are sent over the players headphones	
The ***Atlantis OS Satellite*** notifies ***Atlantis OS Mission Control*** that it has received the message and that the game is now over. If for some reason the game were to end prematurely it would also notified ***Atlantis OS Mission Control*** that fact.	GAME_OVER	
Atlantis OS Mission Control would then communicate with the debriefing station to allow it to play the game and a similar fashion as this example.	DEBRIEF_GAME	Game Number

TABLE II

Console initialized.
Winsock TCP/IP Initialized
WIPX_Init: Unable to open control socket
Exe: 09:30:49 Mar 21 1997
16.0 megabyte heap
Sound Initialization
637k surface cache
320x240
CD Audio Initialized
joystick not found -- no valid joysticks (a5)
execing quake.rc
execing default.cfg
Unknown command "volume"
execing config.cfg
execing autoexec.cfg
8 demo(s) in loop
Playing demo from dontdel.dem.
ERROR: couldn't open.
execing server.cfg
"skill" is "1"
3 demo(s) in loop
VERSION 1.09 SERVER (21066 CRC)

Azure Agony
execing server.cfg
"skill" is "1.000000"
VERSION 1.09 SERVER (21066 CRC)

Azure Agony
Guest313 entered the game
Guest312 entered the game
Debug entered the game
Camera running
I Am Player DragonMan
Guest313 killed Guest312
Guest312 killed Guest312
NET_GetMessage: disconnected socket
VERSION 1.09 SERVER (21066 CRC)

Azure Agony
Camera running
Guest313 entered the game
Guest312 entered the game
Debug entered the game
Guest312 left the game with 0 frags
Guest313 left the game with 0 frags
Camera deactivated
Debug left the game with 0 frags
execing server.cfg
"skill" is "1.000000"
VERSION 1.09 SERVER (21066 CRC)

Azure Agony

Guest313 entered the game
Guest312 entered the game
Debug entered the game
Camera running
NET_GetMessage: disconnected socket
VERSION 1.09 SERVER (21066 CRC)

Azure Agony
Camera running
Guest313 entered the game
Guest312 entered the game
Debug entered the game
Guest313 left the game with 0 frags
Camera deactivated
Guest312 left the game with 0 frags
Debug left the game with 0 frags

TABLE III

What the Atlantis OS Satellite looks for	What Atlantis OS Satellite converts this to
VERSION	*If a player is connected then this is signaling that the game has started.*
I Am Player	*Player has connected to the game*
Playing demo	*System is playing a demo and unless it is the Debriefing machine the system is Ready*
Exited the level	*The player exited the current level and therefore a new level will start. The system will start the next level up with the correct time remaining.*
GameOver	*The game is over*

TABLE IV

Game	Command	Key to Send to Game
Unreal Tournament	Jump	<SPACE>
Unreal Tournament	Fire	<CTRL>
Quake	Jump	<ALT>
Quake 3	Change Weapons	\
All Games	Move Forward	<UP ARROW>
All Games	Move Backward	<DOWN ARROW>

CLAIMS:

1. A method for operating three-dimensional (3D) application software intended to provide a display output to a two-dimensional (2D) screen display, characterized by the steps of:

(a) running the application software in its normal mode to generate 3D application data output which is normally to be sent to an application programming interface (API) driver for the 2D screen display;

(b) intercepting the 3D application data output from the application software and redirecting the data to a pseudo driver for generating a 3D stereoscopic display; and

(c) using the pseudo 3D display driver to generate a 3D stereoscopic display.

2. A method according to Claim 1, wherein the 3D stereoscopic display is selected from the group consisting of head-mounted "stereo vision" goggles, head-mounted 3D display device, and a stereo vision monitor.

3. A method according to Claim 1, wherein the 3D application software is a 3D video game software which provides 3D game data output.

4. A method according to Claim 3, wherein the intercepting and redirecting of the 3D game data is obtained by providing a wrapper for the game software's native API having stereoscopic display function calls linked under the same name as the game software's native API for 2D display.

5. A method according to Claim 1, wherein the pseudo driver generates a 3D stereoscopic display using separate graphics cards or one graphics card with dual heads for rendering right and left image viewpoints for the 3D stereoscopic display.

6. A method according to Claim 3, wherein the intercepted 3D game data is stored in a 3D data recorder for later play back.

7. A method according to Claim 3, wherein the intercepted 3D game data is combined with other 3D content using a mixer and a dual rendering system.

8. A method according to Claim 3, wherein the pseudo 3D display driver is operated with other pseudo drivers such as a stereo sound or a directional force feedback driver.

9. A mission control (administration) system for controlling multiple game playing satellite computers on a network, characterized by:

(a) a mission control computer (10) which operates administrative programs for performing administrative functions for multiple game playing stations connected by the network;

(b) a plurality of game playing satellite computers (20) provided at respective game playing stations each maintaining a plurality of game programs;

(c) a network connecting the mission control computer to the plurality of game playing satellite computers,

(d) wherein said mission control computer includes a mission control program for controlling the plurality of games available to be played on the game playing satellite computers by issuing generic control commands to the game playing satellite computers, and

(e) wherein each of said game playing satellite computers includes a satellite game control program for controlling each of the plurality of game programs available to be played on the satellite computer by receiving a generic control command to start a selected game program issued by said mission control computer and loading in response thereto a game-specific command set corresponding to the selected game program, and by providing said mission control computer with a status report of the status of the selected game program being played on the satellite computer.

10. A system according to Claim 9, wherein a game program on a satellite computer generates one or more of the following sources of information tracking the operation of the game program, and said satellite game control program parses the source of information for desired status information and provides the status information to the mission control program: game log files; dialog boxes or windows opened by the game program; messages from the Notification API; and a method used by the game program for external communications.

11. A system according to Claim 9, wherein the satellite game control program maintains a database of game-specific command sets for each of the game programs offered on the satellite computer, and, when a control command is issued by the mission control computer to start a particular game, the satellite control program loads the corresponding game-specific command set from its database.

12. A system according to Claim 9, wherein said mission control program maintains a database of game data based upon information provided by the game playing satellite computers, and generates one or more administrative reports from the group consisting of: system-wide gaming reports; membership and player statistics; detailed statistics on specific games played by specific players; current status of the system, hardware, and software troubleshooting.

13. A system according to Claim 9, wherein a plurality of mission control computers are maintained at respective mission control sites and are connected via a network to a network server that provides an online interface of the mission control system to the Internet.

14. A system according to Claim 13, wherein said online interface allows players to perform one or more activities of the group consisting of: looking up statistics for games they have played; seeing how their buddies are doing; seeing statistics for comparison at other sites; downloading statistics for their own later use; maintaining their accounts; joining or maintaining their status with a group of players; and communicating with other players.

15. A method for controlling multiple game playing satellite computers on a network, characterized by the steps of:

(a) providing a mission control computer for performing administrative functions for multiple game playing stations on the network;

(b) providing satellite game playing computers at respective game playing stations, each of which maintains a plurality of game programs;

(c) providing a mission control program on the mission control computer for issuing generic control commands to the game playing satellite computers, and

(d) providing a satellite game control program on each of the game playing satellite computers for receiving a generic control command to start a selected game program issued by the mission control computer and loading in response thereto a game-specific command set corresponding to the selected game program, and for providing the mission control computer with a status report of the status of the selected game program being played on the satellite computer.

FIG. 1A

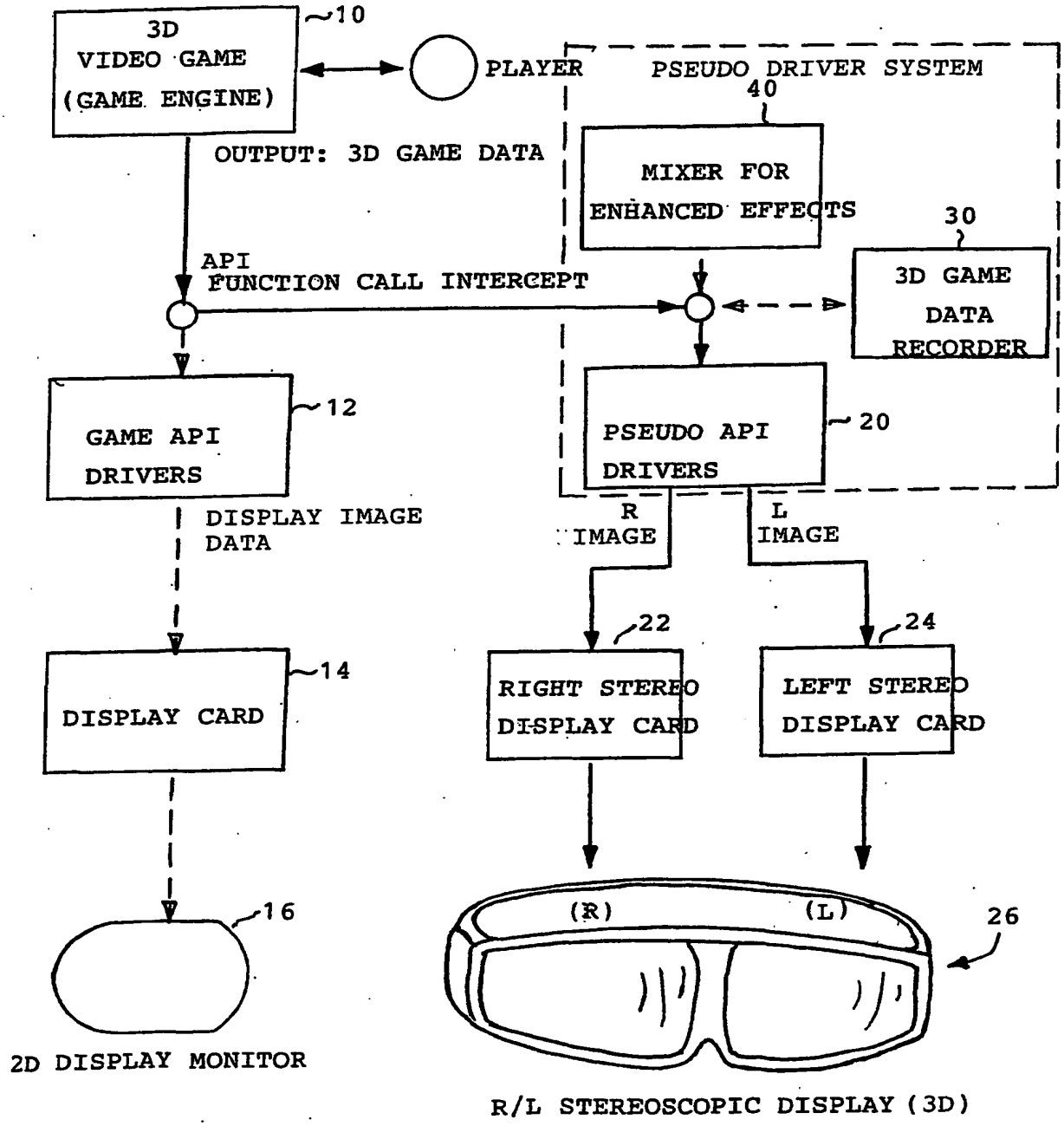

3D VIDEO GAME (GAME ENGINE)
10
PLAYER
PSEUDO DRIVER SYSTEM
40
MIXER FOR ENHANCED EFFECTS
30
3D GAME DATA RECORDER
OUTPUT: 3D GAME DATA
API FUNCTION CALL INTERCEPT
GAME API DRIVERS
12
PSEUDO API DRIVERS
20
DISPLAY IMAGE DATA
R IMAGE
L IMAGE
22
24
DISPLAY CARD
14
RIGHT STEREO DISPLAY CARD
LEFT STEREO DISPLAY CARD
(R)
(L)
26
16
2D DISPLAY MONITOR
R/L STEREOSCOPIC DISPLAY (3D)

FIG. 1B

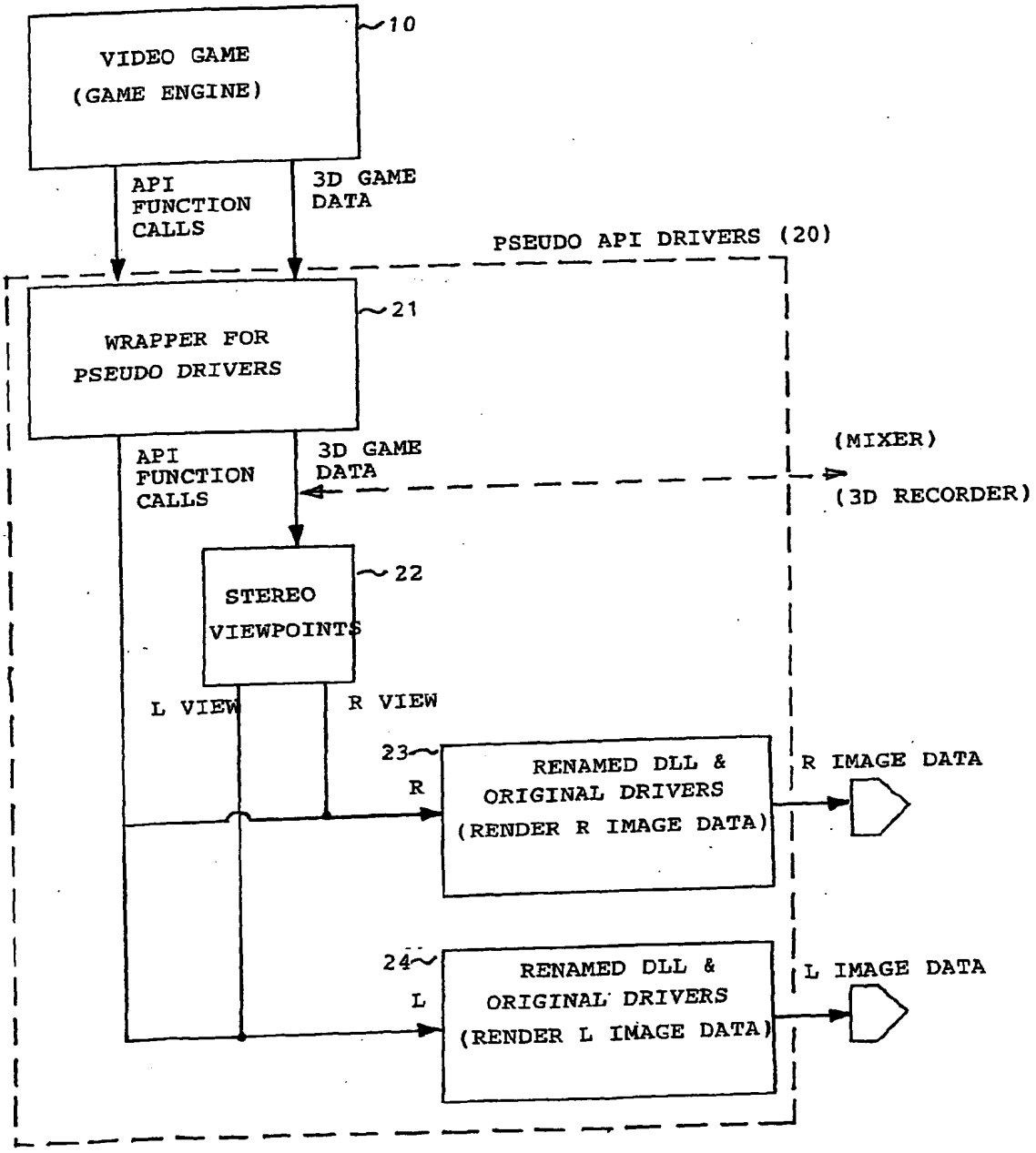

VIDEO GAME
(GAME ENGINE)
10
API
FUNCTION
CALLS
3D GAME
DATA
PSEUDO API DRIVERS (20)
WRAPPER FOR
PSEUDO DRIVERS
21
API
FUNCTION
CALLS
3D GAME
DATA
(MIXER)
(3D RECORDER)
STEREO
VIEWPOINTS
22
L VIEW
R VIEW
23
R
RENAMED DLL &
ORIGINAL DRIVERS
(RENDER R IMAGE DATA)
R IMAGE DATA
24
L
RENAMED DLL &
ORIGINAL DRIVERS
(RENDER L IMAGE DATA)
L IMAGE DATA

FIG. 2A
(PRIOR ART)

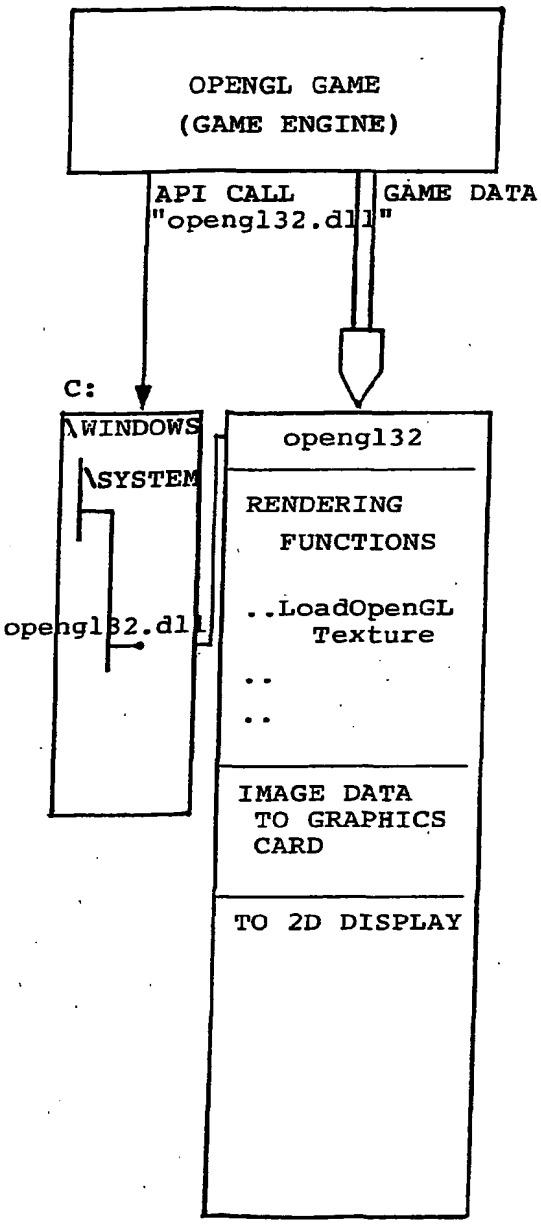


FIG. 2B

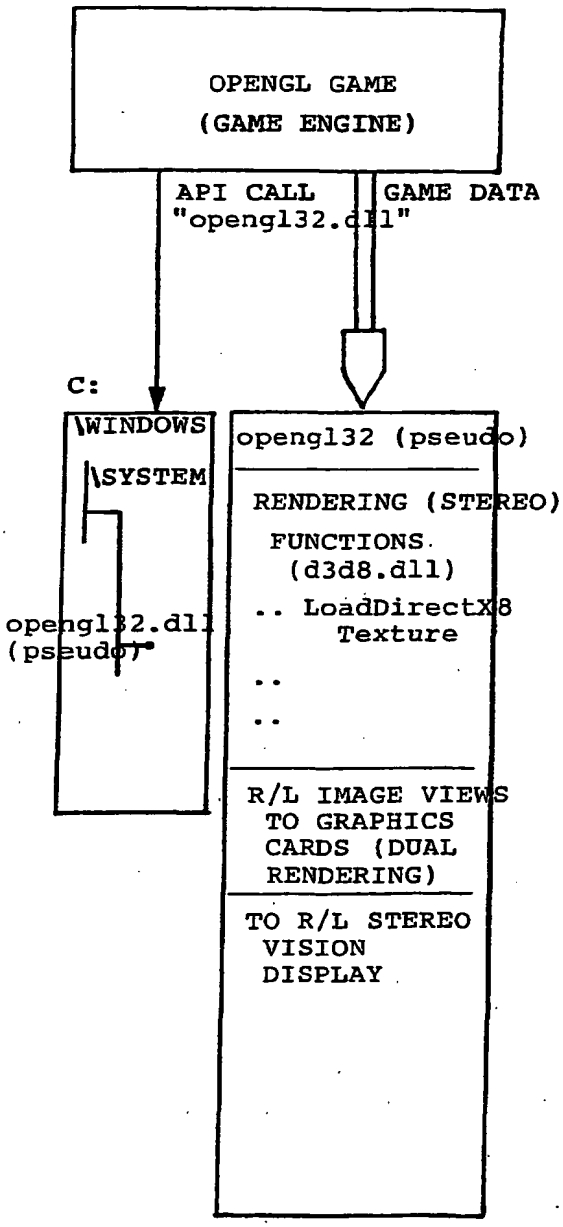


FIG. 3A
(PRIOR ART)

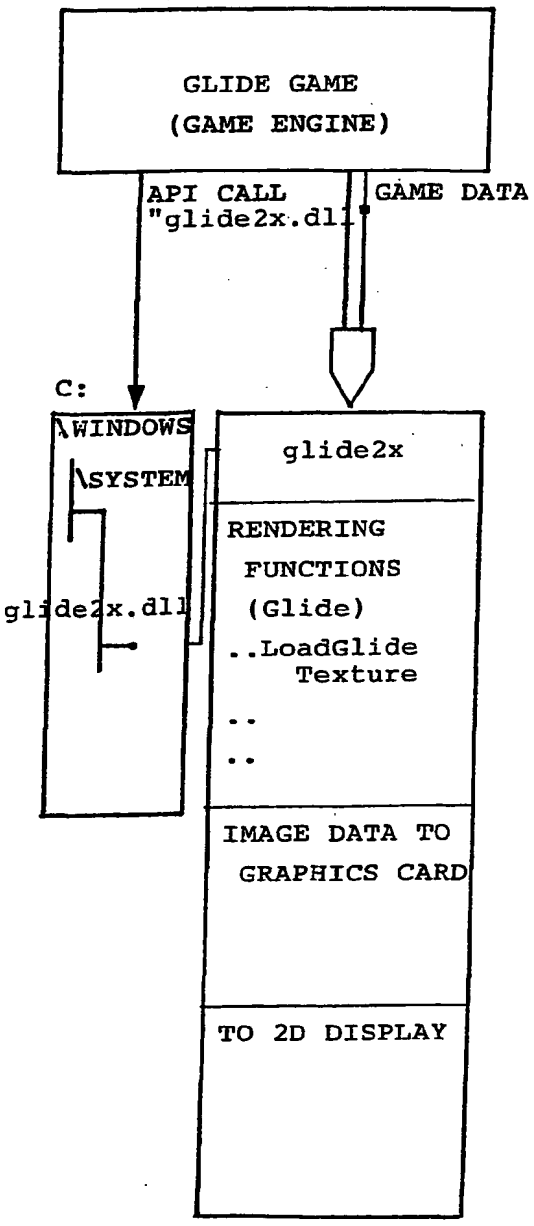


FIG. 3B

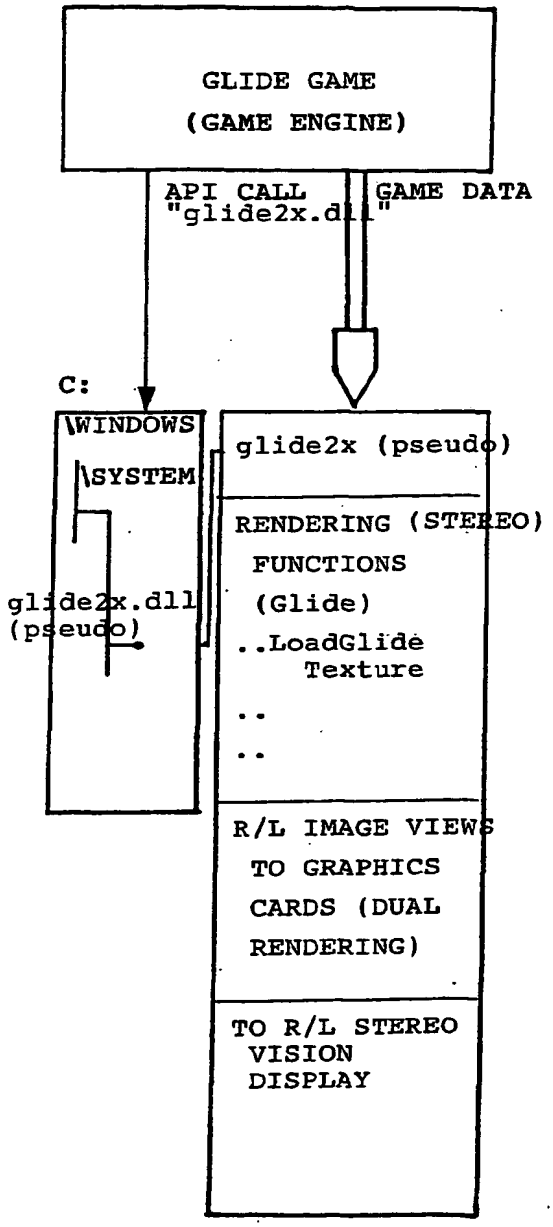


FIG. 4A
(PRIOR ART)

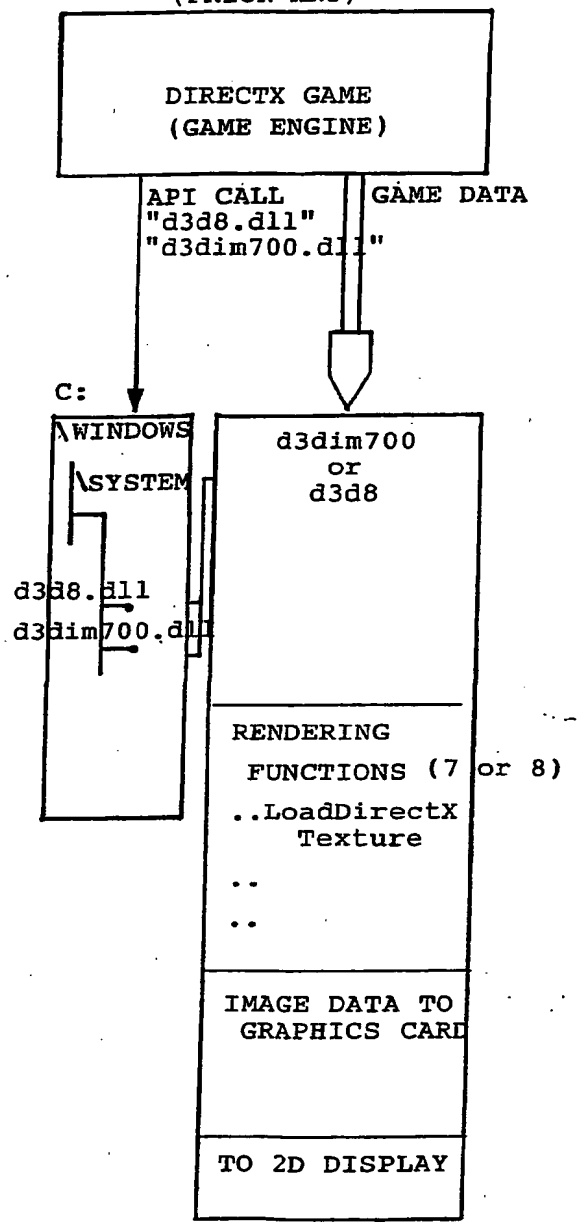


FIG. 4B

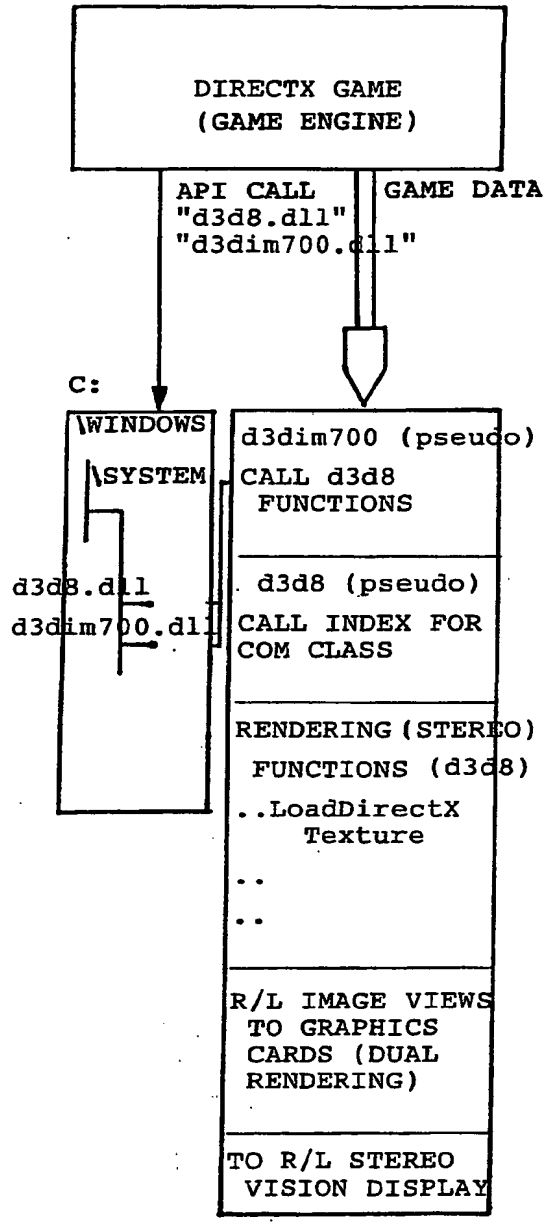


FIG. 5

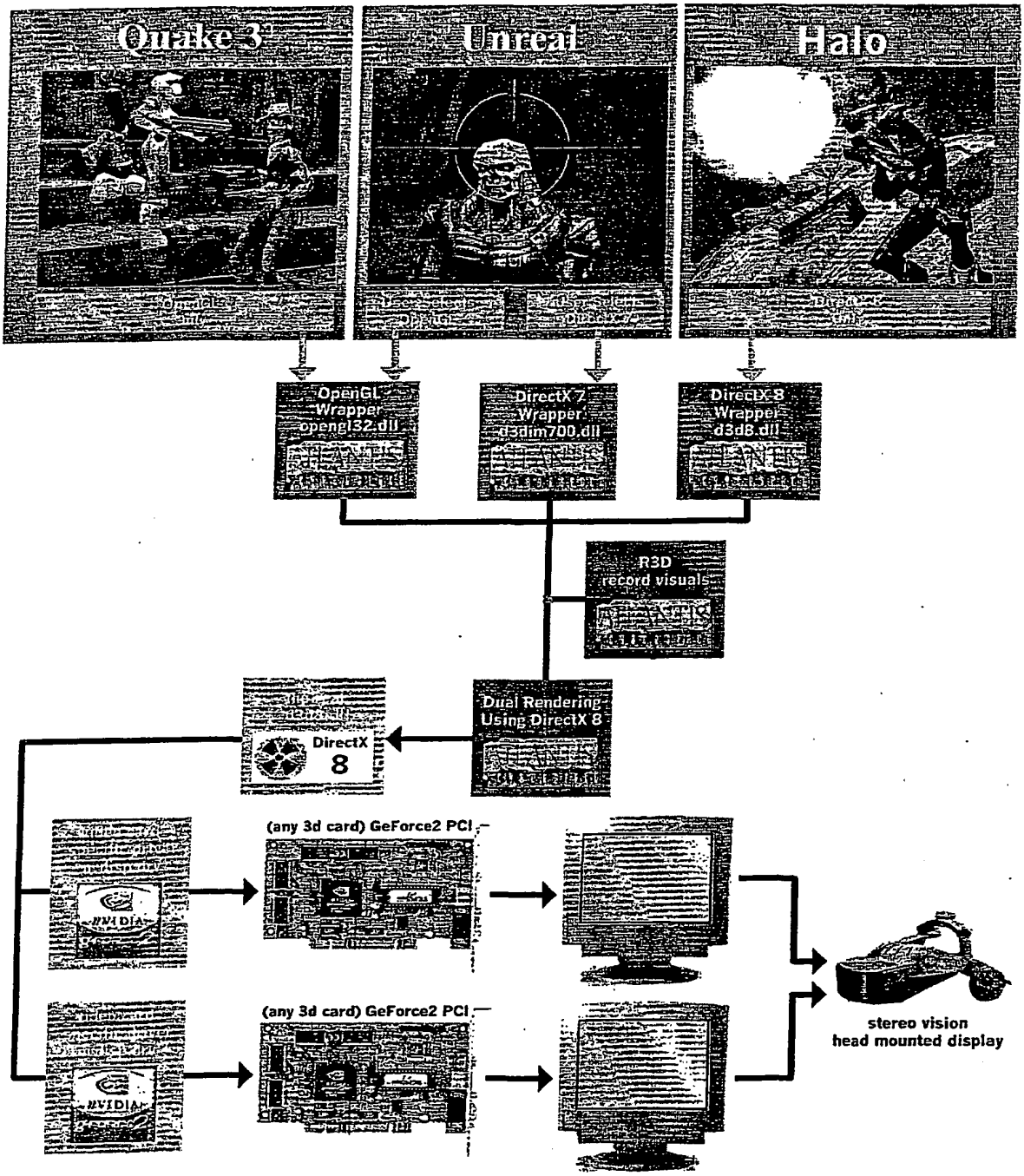
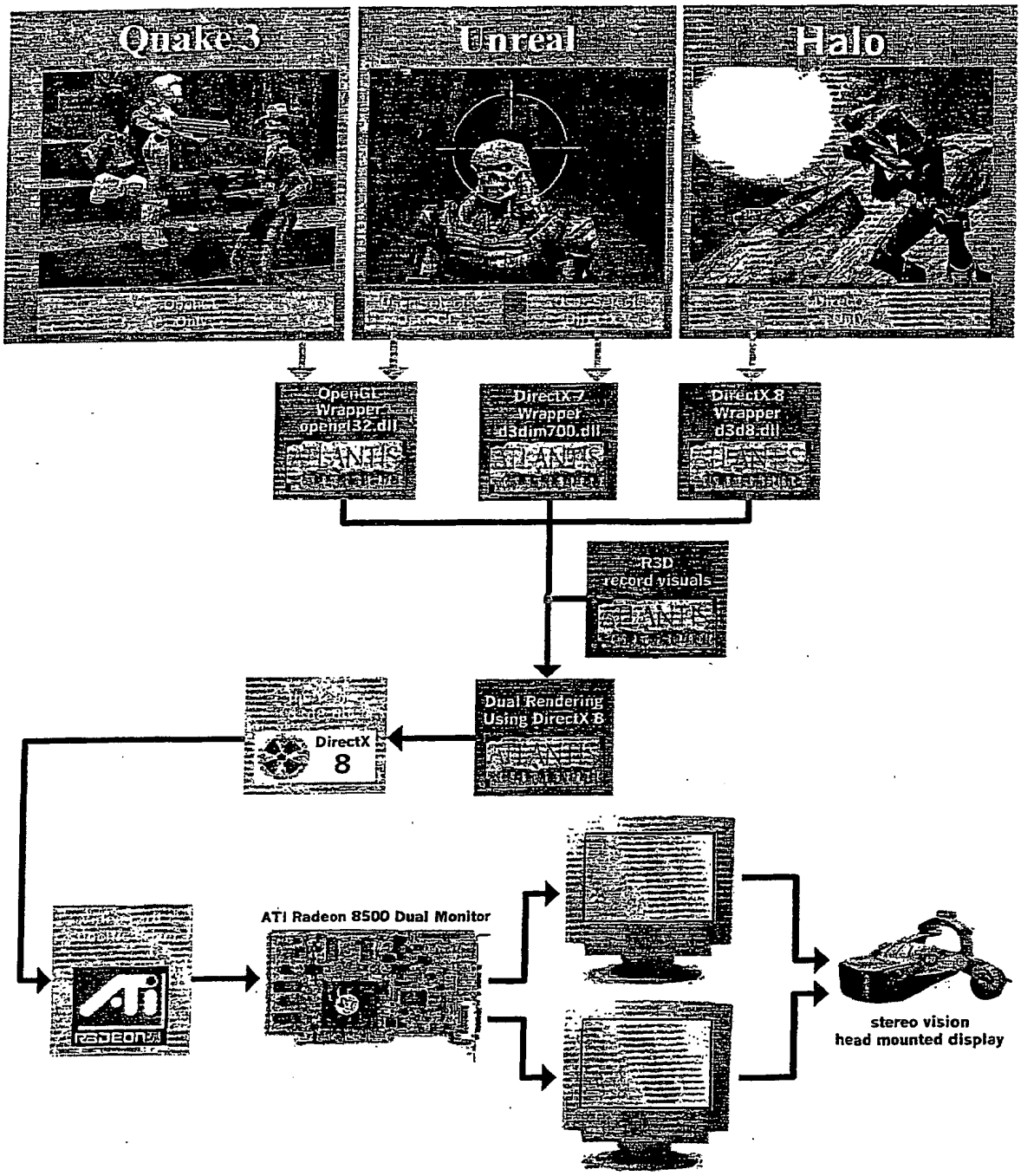

Quake 3
Unreal
Halo
OpenGL Wrapper opengl32.dll
DirectX 7 Wrapper d3dim700.dll
DirectX 8 Wrapper d3d8.dll
R3D record visuals
Dual Rendering Using DirectX 8
DirectX 8
(any 3d card) GeForce2 PCI
(any 3d card) GeForce2 PCI
stereo vision head mounted display

FIG. 6


Quake 3
Unreal
Halo
OpenGL Wrapper opengl32.dll
DirectX 7 Wrapper d3dim700.dll
DirectX 8 Wrapper d3d8.dll
R3D record visuals
Dual Rendering Using DirectX 8
DirectX 8
ATI Radeon 8500 Dual Monitor
stereo vision head mounted display

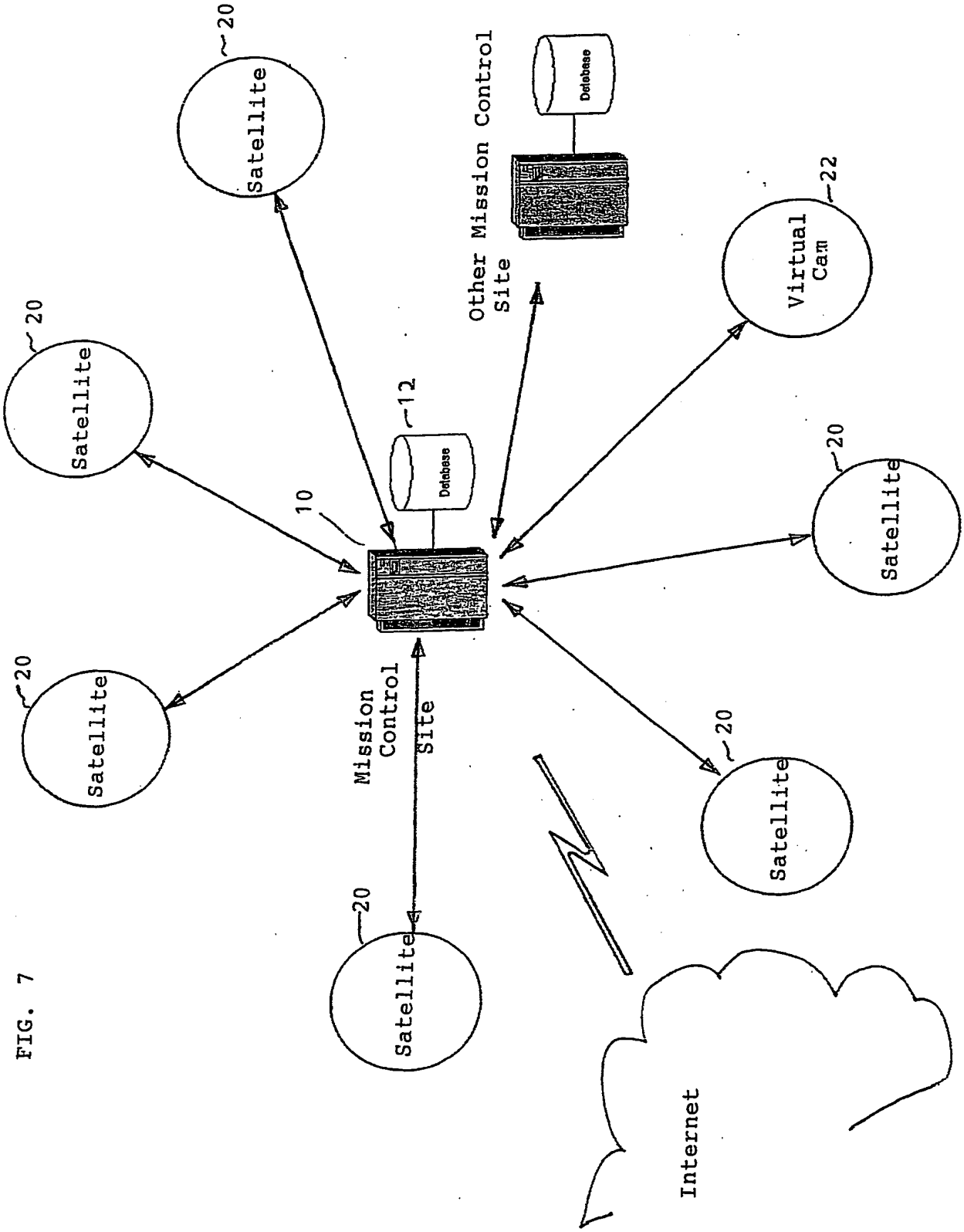

Satellite
20
Satellite
20
Satellite
20
Satellite
20
Satellite
20
Satellite
20
Satellite
20
Other Mission Control Site
Database
Virtual Cam
22
12
Database
10
Mission Control Site
Internet

FIG. 7

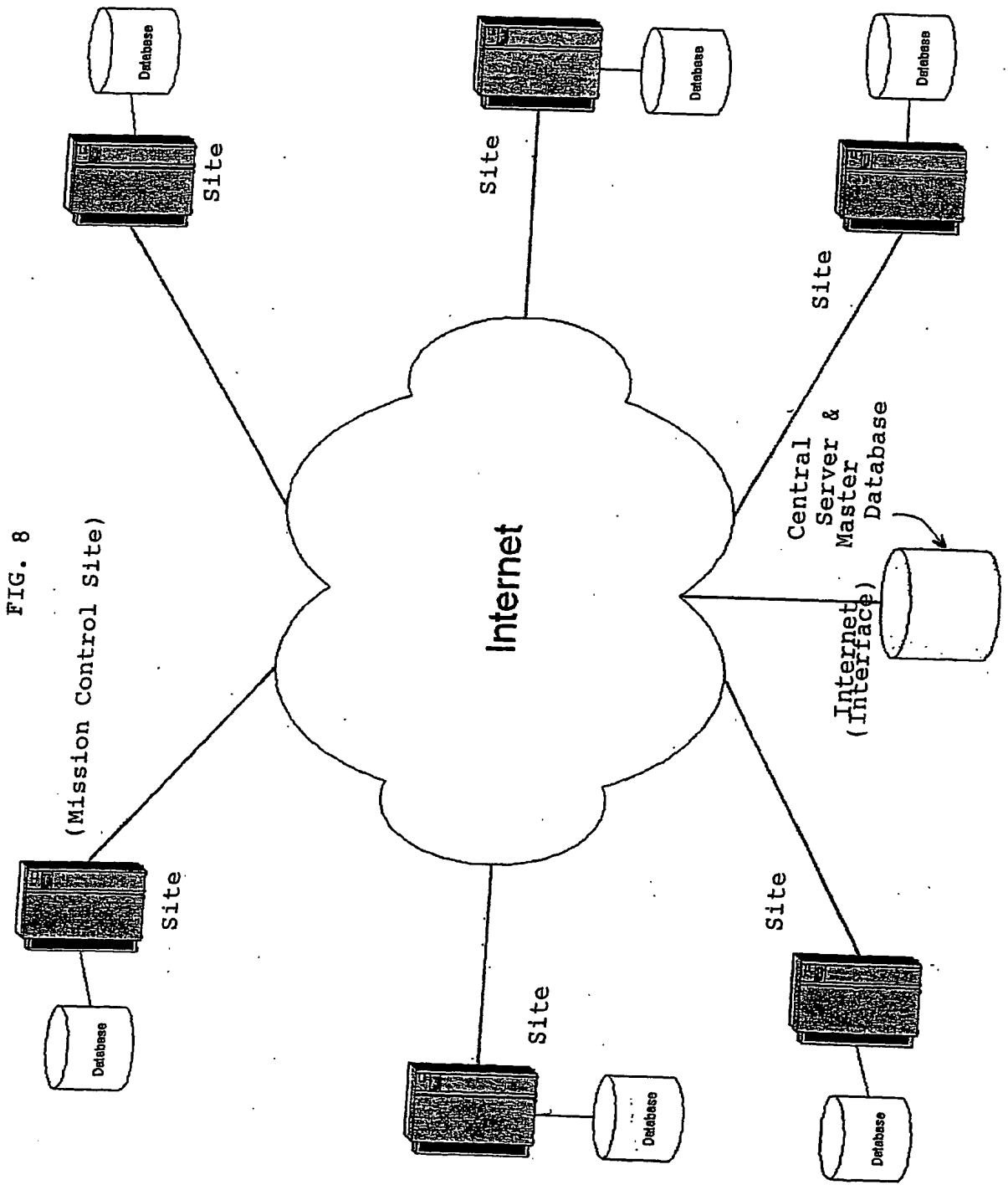

Database
Site
Database
Site
Database
Site
Internet
Central Server & Master Database
(Internet Interface)
(Mission Control Site)
Site
Database
Site
Database
Site
Database

FIG. 8

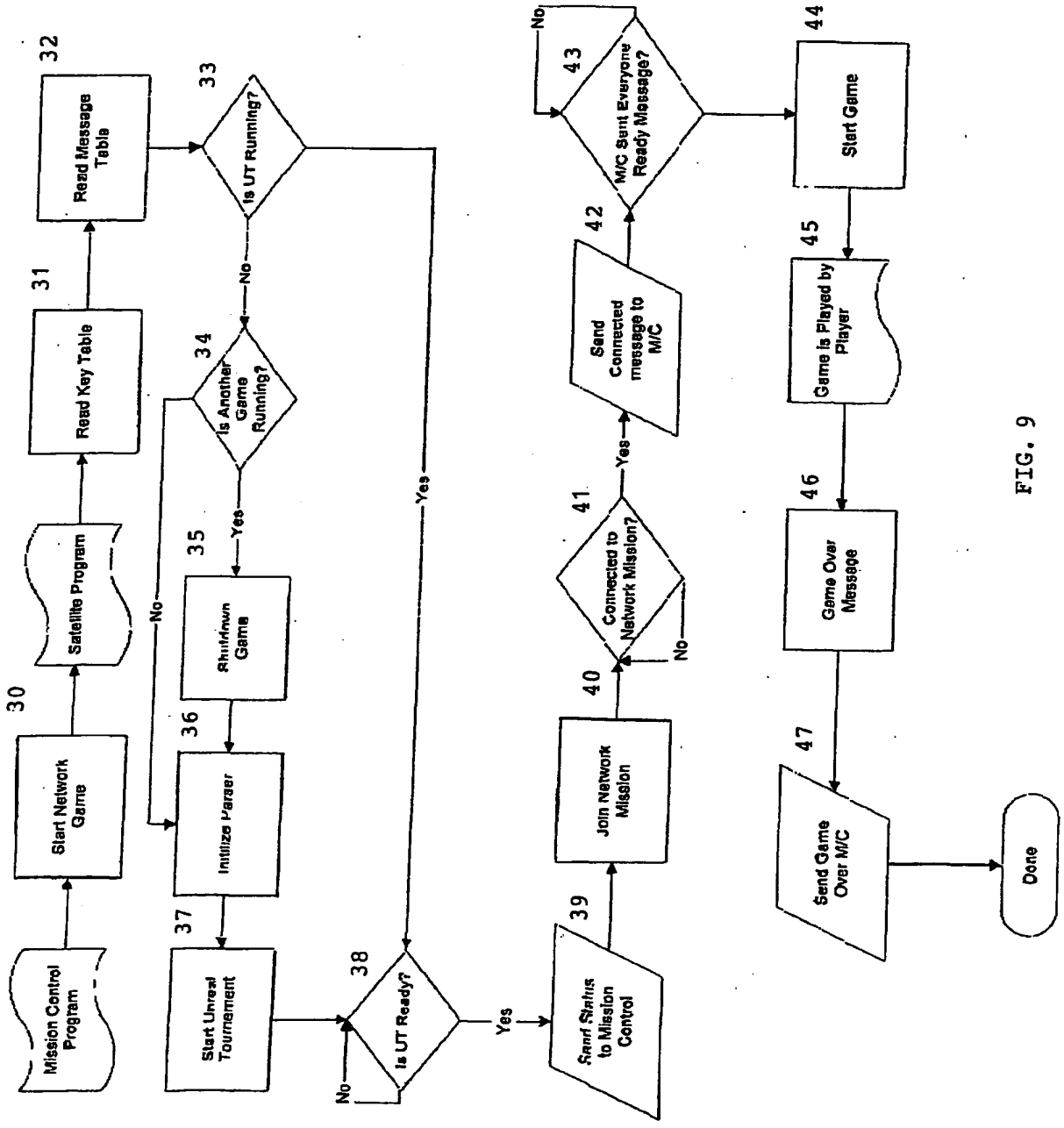

Mission Control Program
30
Start Network Game
Satellite Program
31
Read Key Table
32
Read Message Table
33
Is UT Running?
No
34
Is Another Game Running?
No
Yes
35
Shutdown Game
36
Initialize Parser
37
Start Unreal Tournament
38
Is UT Ready?
No
Yes
Yes
39
Send Status to Mission Control
40
Join Network Mission
41
Connected to Network Mission?
No
Yes
42
Send Connected message to M/C
43
M/C Sent Everyone Ready Message?
No
44
Start Game
45
Game is Played by Player
46
Game Over Message
47
Send Game Over M/C
Done

FIG. 9

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US01/46939

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) :A63F 9/24
US CL :463/32

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 463/32, 29, 40, 41, 42, 43, 44; 345/6, 419

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

Please See Extra Sheet.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A,P	US 6,295,068 B1 (PEDDADA et al) 25 September 2001, see entire document	1-8
A	US 5,796,373 A (MING-YEN) 18 August 1998, see entire document	1-8
A	US 6,099,408 (SCHNEIER et al) 08 August 2000, see entire document.	9-15

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search
23 FEBRUARY 2002

Date of mailing of the international search report
15 MAR 2002

Name and mailing address of the ISA/US
Commissioner of Patents and Trademarks
Box PCT
Washington, D.C. 20231
Facsimile No. (703) 305-3230

Authorized officer
KIM NGUYEN
Telephone No. (703) 308-7[illegible]

Sheila Veney
Paralegal Specialist
Technology Center 3700

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US01/46939

B. FIELDS SEARCHED
Electronic data bases consulted (Name of data base and where practicable terms used):

EAST
search terms: virtual reality, stereoscopic display, three-dimensional, network, server, game stations, game control program, database.